CLAIMS

What is claimed is:

1. A method for arithmetic expression optimization, comprising:

   validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

   optimizing said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said optimizing based at least in part on the relative size of said first type and said second type; and

   matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

2. The method of claim 1 wherein said first instruction is arithmetic.

3. The method of claim 1 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

4. The method of claim 1, further comprising repeating said validating, said optimizing and said matching for instructions that comprise a program.

5. The method of claim 1, further comprising linking each instruction to input instructions in all control paths.

6. The method of claim 1 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

7. The method of claim 6 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

8. The method of claim 6 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

9. The method of claim 1 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further comprising a first input stack and a second input stack; and

said validating further comprises comparing operand types of corresponding entries in said first input stack and said second input stack.

10. The method of claim 9 wherein said comparing further comprises indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

11. The method of claim 1 wherein said optimizing further comprises:

setting said second type to a smallest type;

setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

12. The method of claim 11 wherein said smallest type is the smallest type supported by a target processor.

13. The method of claim 11 wherein said smallest type is the smallest type determined during a previous pass of said optimizing.

14. The method of claim 1 wherein

said third instruction is not a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained; and

setting the type of said third instruction to equal said second type.

15. The method of claim 1 wherein

said third instruction comprises a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained;

setting the type of said third instruction to equal said second type; and

repeating said changing for each input instruction of said third instruction.

16. The method of claim 1, further comprising recording conversion results, said recording comprising:

determining potential overflow associated with said second instruction; and

generating an output stack based at least in part on execution of said second instruction.

17. The method of claim 16 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction

propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

18. The method of claim 17 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

19. The method of claim 16 wherein said generating further comprises:

creating a new output stack based at least in part on one of said at least one input stack;

updating said new output stack based at least in part on operation of said second instruction; and

indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

20. A method for arithmetic expression optimization, comprising:

step for validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

step for optimizing said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said optimizing based at least in part on the relative size of said first type and said second type; and

step for matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

21. The method of claim 20 wherein said first instruction is arithmetic.

22. The method of claim 20 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

23. The method of claim 20, further comprising step for repeating said validating, said optimizing and said matching for instructions that comprise a program.

24. The method of claim 20, further comprising step for linking each instruction to input instructions in all control paths.

25. The method of claim 20 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

26. The method of claim 25 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

27. The method of claim 25 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

28. The method of claim 20 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further comprising a first input stack and a second input stack; and

said validating further comprises comparing operand types of corresponding entries in said first input stack and said second input stack.

29. The method of claim 28 wherein said step for comparing further comprises step for indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

30. The method of claim 20 wherein said optimizing further comprises:

step for setting said second type to a smallest type;

step for setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

step for setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

31. The method of claim 30 wherein said smallest type is the smallest type supported by a target processor.

32. The method of claim 30 wherein said smallest type is the smallest type determined during a previous pass of said optimizing.

33. The method of claim 20 wherein

said third instruction is not a source instruction; and

said step for changing further comprises:

step for recursively examining input instructions until said third instruction is obtained; and

step for setting the type of said third instruction to equal said second type.

34. The method of claim 20 wherein

said third instruction comprises a source instruction; and

said step for changing further comprises:

step for recursively examining input instructions until said third instruction is obtained;

step for setting the type of said third instruction to equal said second type; and

step for repeating said changing for each input instruction of said third instruction.

35. The method of claim 20, further comprising step for recording conversion results, said recording comprising:

step for determining potential overflow associated with said second instruction; and

step for generating an output stack based at least in part on execution of said second instruction.

36. The method of claim 35 wherein said step for determining further comprises:

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if

said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

37. The method of claim 36 wherein said step for determining further comprises:

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

step for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

38. The method of claim 35 wherein said step for generating further comprises:

step for creating a new output stack based at least in part on one of said at least one input stack;

step for updating said new output stack based at least in part on operation of said second instruction; and

step for indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

39. A program storage device readable by a machine, embodying a program of instructions executable by the machine to perform a method for arithmetic expression optimization, the method comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

optimizing said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said optimizing based at least in part on the relative size of said first type and said second type; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

40. The program storage device of claim 39 wherein said first instruction is arithmetic.

41. The program storage device of claim 39 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

42. The program storage device of claim 39, said method further comprising repeating said validating, said optimizing and said matching for instructions that comprise a program.

43. The program storage device of claim 39, said method further comprising linking each instruction to input instructions in all control paths.

44. The program storage device of claim 39 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

45. The program storage device of claim 44 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

46. The program storage device of claim 44 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

47. The program storage device of claim 39 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further comprising a first input stack and a second input stack; and

said validating further comprises comparing operand types of corresponding entries in said first input stack and said second input stack.

48. The program storage device of claim 47 wherein said comparing further comprises indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

49. The program storage device of claim 39 wherein said optimizing further comprises:

setting said second type to a smallest type;

setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

50. The program storage device of claim 49 wherein said smallest type is the smallest type supported by a target processor.

51. The program storage device of claim 49 wherein said smallest type is the smallest type determined during a previous pass of said optimizing.

52. The program storage device of claim 39 wherein

said third instruction is not a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained; and

setting the type of said third instruction to equal said second type.

53. The program storage device of claim 39 wherein

said third instruction comprises a source instruction; and

said changing further comprises:

recursively examining input instructions until said third instruction is obtained;

setting the type of said third instruction to equal said second type; and

repeating said changing for each input instruction of said third instruction.

54. The program storage device of claim 39, further comprising recording conversion results, said recording comprising:

determining potential overflow associated with said second instruction; and

generating an output stack based at least in part on execution of said second instruction.

55. The program storage device of claim 54 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if said second instruction does not create potential overflow, if said second instruction

propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

56. The program storage device of claim 55 wherein said determining further comprises:

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

57. The program storage device of claim 54 wherein said generating further comprises:

creating a new output stack based at least in part on one of said at least one input stack;

updating said new output stack based at least in part on operation of said second instruction; and

indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

58. An apparatus for arithmetic expression optimization, comprising:

means for validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

means for optimizing said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said optimizing based at least in part on the relative size of said first type and said second type; and

means for matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.

59. The apparatus of claim 58 wherein said first instruction is arithmetic.

60. The apparatus of claim 58 wherein said first instruction comprises a non-arithmetic, type-sensitive instruction.

61. The apparatus of claim 58, further comprising means for repeating said validating, said optimizing and said matching for instructions that comprise a program.

62. The apparatus of claim 58, further comprising means for linking each instruction to input instructions in all control paths.

63. The apparatus of claim 58 wherein

said first instruction is defined for a first processor having a first base; and

said second instruction is defined for a second processor having a second base.

64. The apparatus of claim 63 wherein

said first processor comprises a Java™ Virtual Machine; and

said second processor comprises a Java Card™ Virtual Machine.

65. The apparatus of claim 63 wherein

said first processor comprises a 32-bit processor; and

said second processor comprises a resource-constrained 16-bit processor.

66. The apparatus of claim 58 wherein

said at least one input stack comprises a plurality of input stacks, said plurality of input stacks further comprising a first input stack and a second input stack; and

said means for validating further comprises means for comparing operand types of corresponding entries in said first input stack and said second input stack.

67. The apparatus of claim 66 wherein said means for comparing further comprises means for indicating an error if the types of at least said first at least one stack entry and said second at least one stack entry are not equivalent.

68. The apparatus of claim 58 wherein said means for optimizing further comprises:

means for setting said second type to a smallest type;

means for setting said second type to the type of an operand in said at least one input stack if said smallest type is less than said type of said operand; and

means for setting said second type to a type that is larger than said smallest type if said smallest type is greater than said type of said operand, if said operand has potential overflow, if said second instruction is sensitive to overflow and if said second type is less than said first type.

69. The apparatus of claim 68 wherein said smallest type is the smallest type supported by a target processor.

70. The apparatus of claim 68 wherein said smallest type is the smallest type determined during a previous pass of said optimizing.

71. The apparatus of claim 58 wherein

said third instruction is not a source instruction; and

said means for changing further comprises:

means for recursively examining input instructions until said third instruction is obtained; and

means for setting the type of said third instruction to equal said second type.

72. The apparatus of claim 58 wherein

said third instruction comprises a source instruction; and

said means for changing further comprises:

means for recursively examining input instructions until said third instruction is obtained;

means for setting the type of said third instruction to equal said second type; and

means for repeating said changing for each input instruction of said third instruction.

73. The apparatus of claim 58, further means for comprising recording conversion results, said recording comprising:

means for determining potential overflow associated with said second instruction; and

means for generating an output stack based at least in part on execution of said second instruction.

74. The apparatus of claim 73 wherein said means for determining further comprises:

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if said second instruction creates potential overflow; and

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if

said second instruction does not create potential overflow, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

75. The apparatus of claim 74 wherein said determining further comprises:

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, and if overflow is possible based at least in part on the type of said second instruction and the relationship between said first type and said second type; and

means for indicating said second instruction has potential overflow if said second type does not equal said first type, if said second instruction does not remove potential overflow, if overflow is not possible based at least in part on the type of said second instruction and the relationship between said first type and said second type, if said second instruction propagates potential overflow, and if at least one operand in said at least one input stack has potential overflow.

76. The apparatus of claim 73 wherein said means for generating further comprises:

means for creating a new output stack based at least in part on one of said at least one input stack;

means for updating said new output stack based at least in part on operation of said second instruction; and

means for indicating another instruction conversion pass is required if said new stack does not equal a previous output stack.

77. A method of using an application software program including arithmetic expression optimization of at least one instruction targeted to a processor, the method comprising:

receiving the software program on said processor, said software program optimized according to a method comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

optimizing said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said optimizing based at least in part on the relative size of said first type and said second type; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand; and

executing said at least one instruction on said processor.

78. A smart card having a microcontroller embedded therein, said microcontroller configured to execute a virtual machine, the virtual machine capable of executing a software application

comprising a plurality of previously optimized instructions, the instructions optimized by a method comprising:

validating at least one input stack associated with a first instruction configured to operate on at least one operand of a first type, each of said at least one input stack associated with an input instruction of said first instruction, each input stack representing the state of an operand stack associated with an input instruction upon execution of said input instruction;

optimizing said first instruction to a second instruction configured to operate on at least one operand of a second type, said second type smaller than said first type, said optimizing based at least in part on the relative size of said first type and said second type; and

matching said second type with an operand type of at least one operand in said at least one input stack associated with said second instruction, said matching comprising changing the type of instructions in a chain of instructions to equal said second type if said operand type is less than said second type, said chain bounded by said second instruction and a third instruction that is the source of said at least one operand.